REMARKS

Claims 1-25 are pending. Claims 1, 15, 22 and 24 are independent.

Claims 1-5, 8, 11, 15-20 and 22-25 are rejected. Claims 6, 7, 9, 10, 12-14 and 21 are objected to.

The examiner continues to use Moller to reject claims 1, 15, 16, 20, 22 and 24 as having been anticipated.

For a reference to anticipate a claim, each element and limitation of the claim must be found in the reference. Hoover Group, Inc. v. Custom Metalcraft, Inc., 66 F.3d 299, 302 (Fed. Cir. 1995). Claim 1, as amended, recites "directing the processor having a plurality of microengines, each of the microengines maintaining a plurality of program counters in hardware and states associated with the program counters, to swap a currently running context in a specified microengine out to memory to let another context execute in that microengine and cause a different context and associated program counter to be selected." Moller neither describes nor suggests this quoted claim feature.

The examiner expresses confusion over applicant's claimed "microengine" language. Microengines are clearly described in applicant's detailed description:

> **The hardware-based multithreaded processor 12 also includes a plurality of function microengines 22a-22f. Functional microengines (microengines) 22a-22f each maintain a plurality of program counters in hardware and states associated with the program counters. Effectively, a corresponding plurality of sets of threads can be simultaneously active on each of the microengines 22a-22f while only one is actually operating at any one time.**
>
> **In one embodiment, there are six microengines 22a-22f as shown. Each microengines 22a-22f has capabilities for processing four hardware threads. The six microengines 22a-22f operate with shared resources including memory system 16 and bus interfaces 24 and 28. The memory system 16 includes a Synchronous Dynamic Random Access Memory (SDRAM) controller 26a and a Static Random Access Memory (SRAM) controller 26b. SDRAM memory 16a and SDRAM controller 26a are typically used for processing large volumes of data, e.g., processing of network payloads from network packets. The SRAM controller 26b and SRAM memory 16b are used in a networking implementation for low latency, fast access tasks, e.g., accessing look-up tables, memory for the core processor 20, and so forth. (Specification, page 4, lines 4-24)**

Microengines are further described with reference to FIG. 3 and the accompanying text in the detailed description:

> **Referring to FIG. 3, an exemplary one of the microengines 22a-22f, e.g., microengine 22f is shown. The microengine includes a control store 70 which, in one implementation, includes a RAM of here 1,024 words of 32 bit. The RAM stores a microprogram. The microprogram is loadable by the core processor 20. The microengine 22f also includes controller logic 72. The controller logic includes an instruction decoder 73 and program counter (PC) units 72a-72d. The four micro program counters 72a-72d are maintained in hardware. The microengine 22f also includes context event switching logic 74. Context event logic 74 receives messages (e.g., SEQ_#_EVENT_RESPONSE; FBI_EVENT_RESPONSE; SRAM _EVENT_RESPONSE; SDRAM _EVENT_RESPONSE; and ASB _EVENT_RESPONSE) from each one of the shared resources, e.g., SRAM 26a, SDRAM 26b, or processor core 20, control and status registers, and so forth. These messages provide information on whether a requested function has completed. Based on whether or not a function requested by a thread has completed and signaled completion, the thread needs to wait for that completion signal, and if the thread is enabled to operate, then the thread is placed on an available thread list (not shown). The microengine 22f can have a maximum of e.g., 4 threads available.**
>
> **In addition to event signals that are local to an executing thread, the microengines 22 employ signaling states that are global. With signaling states, an executing thread can broadcast a signal state to all microengines 22. Receive Request Available signal, Any and all threads in the microengines can branch on these signaling states. These signaling states can be used to determine availability of a resource or whether a resource is due for servicing.**
>
> **The context event logic 74 has arbitration for the four (4) threads. In one embodiment, the arbitration is a round robin mechanism. Other techniques could be used including priority queuing or weighted fair queuing. The microengine 22f also includes an execution box (EBOX) data path 76 that includes an arithmetic logic unit 76a and general purpose register set 76b. The arithmetic logic unit 76a performs arithmetic and logical functions as well as shift functions. The registers set 76b has a relatively large number of general purpose registers. As will be described in FIG. 6, in this implementation there are 64 general purpose registers in a first bank, Bank A and 64 in a second bank, Bank B. The general purpose registers are windowed as will be described so that they are relatively and absolutely addressable.**
>
> **The microengine 22f also includes a write transfer register stack 78 and a read transfer stack 80. These registers are also windowed so that they are relatively and absolutely addressable. Write transfer register stack 78 is where write data to a resource is located. Similarly, read register stack 80 is for return data from a shared resource. Subsequent to or concurrent with data arrival, an event signal from the respective shared resource e.g., the SRAM controller 26a, SDRAM controller 26b or core processor 20 will be provided to context event arbiter 74 which will then alert the thread that the data is available or has been sent. Both transfer register banks 78 and 80 are connected to the execution box (EBOX) 76 through a data path. In one implementation, the read transfer register has 64 registers and the write transfer register has 64 registers. (Specification page 14, line 21 to page 17, line 7)**

The examiner also expresses concern with respect to swapping. Here again, applicant's claimed invention is fully described in the detailed description:

By employing hardware context swapping within each of the microengines 22a-22f, the hardware context swapping enables other contexts with unique program counters to execute in that same microengine. (Specification page 5, lines 17-20)

Applicant's detailed description describes instructions used in the invention:

The context swap instruction CTX_ARB swaps a currently running context in a specified microengine out to memory to let another context execute in that microengine. The context swap instruction CTX_ARB also wakes up the swapped out context when a specified signal is activated. The format for the context swap instruction is:

ctx_arb[parameter], optional_token

The "parameter" field can have one of several values. If the parameter is specified as "sram Swap", the context swap instruction will swap out the current context and wake it up when the thread's SRAM signal is received. If the parameter is specified as "sram Swap", the context swap instruction will swap out the current context and wake it up when the thread's SDRAM signal is received. The parameter can also be specified as "FBI" and swap out the current context and wake it up when the thread's FBI signal is received. The FBI signal indicates that an FBI CSR, Scratchpad, TFIFO, or RFIFO operation has completed.

The parameter can also be specified as "seq_num1_change/seq_num2_change", which swaps out the current context and wakes it up when the value of the sequence number changes. The parameter can be specified as "inter_thread" which swaps out the current context and wakes it up when the threads interthread signal is received, or "voluntary" which will swap out the current context if another thread is ready to run, otherwise do not swap. If the thread is swapped, it is automatically re-enabled to run at some subsequent context arbitration point. The parameter can be "auto_push" which swaps out the current context and wakes it up when SRAM transfer read register data has been automatically pushed by the FBus interface, or a "start_receive" that swaps out the current context and wake it up when new packet data in the receive FIFO is available for this thread to process.

The parameter can also be "kill" which prevents the current context or thread from executing again until the appropriate enable bit for the thread is set in a CTX_ENABLES register, "pci" which swaps out the current context and wake it up when the PCI unit signals that a DMA transfer has been completed.

The context swap instruction CTX_ARB can have the following optional_token, defer one which specifies that one instruction will be executed after this reference before the context is swapped. (Specification page 28, line 15 to page 30, line 8)

In summary, the microengines support up to four threads in hardware. A programmer can switch between these threads with no overhead by issuing a context swap instruction, which is often used in conjunction with memory accesses or thread signaling. There is hardware support for queues in SRAM. These queues are accessed by "push" and "pop" instructions. There is interrupt-driven execution for the threads. In other words, a thread can swap itself out and only restart once it has received a signal from another thread, memory, or the processor.

Moller discloses a processor. Moller's processor does not include multiple microengines. Moller discloses a swap instruction:

> **Execution of the SWAP microinstruction causes CONTROL 100 to generate a CMUXCTL signal which is applied to C-MUX 118 causing it to pass the top of LIFO stack 136 contents on signal lines 140 to the down counter 120. At the same CONTROL 100 generates a STKMUXCTL signal which applied to STACK MUX 132 causes the STACK MUX to select the signal on lines 122 from the down counter 120 to be read onto the top of stack 136 upon receipt of the WE signal by stack 136 as subsequently generated by CONTROL 100. CONTROL 100 does not, under these circumstances, generate an UP signal so that the current top of stack contents are overwritten, having been loaded into down counter 120, so that the iteration count for the invoking loop replaces the iteration count for the invoked loop. (Col. 30, lines 39-54)**

As clearly seen, Moller's swap instruction is very different from directing the processor having a plurality of microengines, each of the microengines maintaining a plurality of program counters in hardware and states associated with the program counters, to swap a currently running context in a specified microengine out to memory to let another context execute in that microengine and cause a different context and associated program counter to be selected. Accordingly, claim 1 is not anticipated by Moller.

Moller discloses traditional stack operations in conjunction with its swap microinstruction. Moller discloses the receipt of signals. Claims 15, 22 and 25, as amended, recite "performing a swapping operation to cause a different context and associated program counter to be selected in accordance with the value of the evaluated specified parameter," or similar langiage. Applicant's specified parameter represents a state of an executing context process. Moller neither describes nor suggests this quoted claim feature. More specifically, Moller fails to describe or suggest causing any different context and associated program counter to be selected in accordance with the value of the evaluated specified parameter. Accordingly, claims 15, 22 and 25 are not anticipated by Moller.

The examiner uses Moller and Adkins to reject claims 2, 3, 8, 17, 23 and 25 as having been obvious.

Claims 1, 15, 22 and 24 are not rendered obvious by Moller and Adkins. Moller was discussed above. Claim 1 recites "directing the processor having a plurality of microengines, each of the microengines maintaining a plurality of program counters in hardware and states

associated with the program counters, to swap a currently running context in a specified microengine out to memory to let another context execute in that microengine and cause a different context and associated program counter to be selected." Claims 15, 22 and 24 recite "causing any different context and associated program counter to be selected in accordance with the value of the evaluated specified parameter." Adkins fails to teach or suggest these quoted claim features. Adkins teaches:

> **(A) serial communications adapter provides an interface to physical communications ports. A scheduler executing on the adapter schedules tasks at different priority levels, so that time-critical tasks are performed quickly enough to prevent data loss. Data to be transmitted or received through a communications port is stored in buffers on the adapter, and data and command communications between the adapter and the host system are preferably performed over a DMA channel. (Col. 1, lines 59-68)**

No combination of Moller and Adkins teaches or suggests these independent claim features. Claims 2, 3, 8, 17, 23 and 25 depend upon, and add further limitations to, claims 1, 15, 22 and 24. Accordingly, claims 2, 3, 8, 17, 23 and 25 are not rendered obvious by Moller and Adkins.

The examiner uses Moller, Adkins and Angle to reject claims 4 and 18 as having been obvious.

Claims 1 and 15 are not rendered obvious by Moller, Adkins and Angle. Moller and Adkins were discussed above. Angle does not teach or suggest "directing the processor having a plurality of microengines, each of the microengines maintaining a plurality of program counters in hardware and states associated with the program counters, to swap a currently running context in a specified microengine out to memory to let another context execute in that microengine and cause a different context and associated program counter to be selected," (Claim 1) or "causing any different context and associated program counter to be selected in accordance with the value of the evaluated specified parameter." (Claim 15) Angle teaches a packet streaming engine that manipulates data contained within a stream of packets using a delayed packet mechanism:

> **The delayed replace mechanism allows for packet streaming where embedded fields within a data packet may be modified dependent upon fields following them within the data packet. (see Abstract)**

No combination of Moller, Adkins and Angle teaches or suggests these independent claim features. Claims 4 and 18 depend upon, and add further limitations to, claims 1 and 15. Accordingly, claims 4 and 18 are not rendered obvious by Moller, Adkins and Angel.

The examiner uses Moller, Adkins and Manning to reject claims 5 and 19 as having been obvious.

Claims 1 and 15 are not rendered obvious by Moller, Adkins and Manning. Moller and Adkins were discussed above. Manning does not teach or suggest "directing the processor having a plurality of microengines, each of the microengines maintaining a plurality of program counters in hardware and states associated with the program counters, to swap a currently running context in a specified microengine out to memory to let another context execute in that microengine and cause a different context and associated program counter to be selected," (Claim 1) or "causing any different context and associated program counter to be selected in accordance with the value of the evaluated specified parameter." (Claim 15) Manning teaches an integrated circuit memory device with a standard DRAM pinout that is designed for high speed data access and for compatibility with existing memory systems. (Col. 3, lines 14-16)

No combination of Moller, Adkins and Manning teaches or suggests these quoted independent claim features. Claims 5 and 19 depend upon, and add further limitations to, claims 1 and 15. Accordingly, claims 5 and 19 are not rendered obvious by Moller, Adkins and Manning.

The examiner uses Moller, Adkins and Turner to reject claim 11 as having been obvious.

Claim 1 is not rendered obvious by Moller, Adkins and Turner. Moller and Adkins were discussed above. Turner does not teach or suggest "directing the processor having a plurality of microengines, each of the microengines maintaining a plurality of program counters in hardware and states associated with the program counters, to swap a currently running context in a specified microengine out to memory to let another context execute in that microengine and cause a different context and associated program counter to be selected." Turner teaches:

> **Non-executing threads exist in one of three states: Ready (40), Suspended (41), and Suspended-Waiting (42). This minimized set of possible thread states further enables efficiency of the embedded processing system. Threads are initially created in the Ready state (40). In the preferred embodiment, Ready threads are linked members of a Ready list,**

> **and are queued and awaiting for some combination of synchronous and asynchronous preemption events to advance in the Ready list and run. (col. 5, lines 56-64)**

Further, Turner uses techniques well known in the art:

> **Control of the processor execution can be transferred from one thread to another by synchronous or asynchronous preemption. This concept is common in the art. Synchronous preemption is based on an event, such as receipt of a hardware interrupt signal, or the expiration of a hardware timer. (Col. 6, lines 38-43)**

No combination of Moller, Adkins and Turner teaches or suggests this independent claim feature. Claim 11 depends upon, and adds further limitations to, claim 1. Accordingly, claim 11 is not rendered obvious by Moller, Adkins and Turner.

It is believed that all of the pending claims have been addressed. However, the absence of a reply to a specific rejection, issue or comment does not signify agreement with or concession of that rejection, issue or comment. In addition, because the arguments made above may not be exhaustive, there may be reasons for patentability of any or all pending claims (or other claims) that have not been expressed. Finally, nothing in this paper should be construed as an intent to concede any issue with regard to any claim, except as specifically stated in this paper, and the amendment of any claim does not necessarily signify concession of unpatentability of the claim prior to its amendment.

Please apply any charges or credits to deposit account 06-1050.

Respectfully submitted,

Date: May 17, 2005

Kenneth F. Kozik
Kenneth F. Kozik
Reg. No. 36,572

ATTORNEYS FOR INTEL CORPORATION
Fish & Richardson P.C.
225 Franklin St.
Boston, MA 02110
Telephone: (617) 542-5070
Facsimile: (617) 542-8906
21089243.doc